# Instance-Based Opponent Action Prediction in Soccer Simulation Using Boundary Graphs

Thomas Gabel and Fabian Sommer

Faculty of Computer Science and Engineering
Frankfurt University of Applied Sciences
60318 Frankfurt am Main, Germany
{tgabel|fsommer}@fb2.fra-uas.de

**Abstract.** The ability to correctly anticipate an opponent's next action in real-time adversarial environments depends on both, the amount of collected observations of that agent's behavior as well as on the capability to incorporate new knowledge into the opponent model easily. We present a novel approach to instance-based action prediction that utilizes graph-based structures for the efficiency of retrieval, that scales logarithmically with the amount of training data, and that can be used in an online and anytime manner. We apply this algorithm to the use case of predicting a dribbling agent's next action in Soccer Simulation 2D.

## 1 Introduction

Opponent modeling and action prediction have a long history in robotic soccer. The ability to anticipate what an opponent player is going to do in the next time step and reacting with appropriate counter measures can bring about significant advantages to one's own agents. In this paper, we extend our previous work [8] on predicting the low-level behavior of agents in Soccer Simulation 2D into a direction that makes it scalable and practically applicable under the hard real-time constraints that are imposed in this domain. Hitherto, we approached the task of predicting an opponent's next action in an instance-based manner by explicitly storing all training instances in memory and then (e.g. during a match) searching linearly for the nearest neighbor to the current situation and using that neighbor's class label as the predicted next opponent action.

Unfortunately, it is well-known that instance-based classification approaches, when being applied in the described naive manner, scale poorly (usually linearly) with the amount of training data. As a consequence, when intending to apply these ideas in our soccer simulation competition team, we arrive at a set of challenging requirements:

a) *Instance-based:* Instance-based learning is a lazy learning approach; new instances shall, if necessary, be memorized easily.
b) *Real-time capable:* There are hard real-time constraints in robotic soccer. Thus, when searching for the nearest neighbor from the set of stored instances, hard time limits must be respected.

c) *Incremental:* The approach should be applicable online, which means that it must be possible to incorporate new experience on the fly during the application without the need to perform some computationally heavy relearning.
d) *Anytime:* Usually, the available computational budget varies from time to time. Thus, it would be highly desirable to have an anytime prediction algorithm whose accuracy improves with more computational power.
e) *Simplicity:* The desired prediction algorithm shall be simple to implement and have no dependency on certain libraries or mighty learning frameworks such that it can be utilized easily on any (competition) machine.

Needless to say that any prediction algorithm with linear time requirements in the number $n$ of training examples is ruled out as it would perform too poorly and not scale for larger dataset sizes. Hence, for an instance-based prediction algorithm ideally logarithmic complexity is desired or at least a dependency on $n$ according to some power law with a power value significantly below one.

Our contribution in this paper is twofold. On the one hand, we propose a novel instance-based classification approach that fulfills the mentioned five requirements. At the heart of this approach is the construction of an index structure to efficiently guide the search for most similar instances that we call a *Boundary Graph*. We build up the graph structure from training data, which means that its topology is not fixed a priori. It is also worth noting that the construction process can be applied in an online setting, i.e. no batch access to the full dataset of instances is needed and, hence, the graph index structure can be extended as more and more training examples come in. Both, the build-up as well as the employment of that graph-based index structure are inherently stochastic – a fact that we found to substantially improve the robustness of the approach as well as to reduce its dependency on other factors like the order of presentation of instances during learning.

On the other hand, we empirically evaluate the performance of the delineated approach for the use case of predicting a dribbling opponent agent's next action in soccer simulation. Knowing the opponent's next action with high certainty before it is executed by the opponent may enable our agents to simultaneously compute the best possible answer to that future action and, hence, improve our team's playing strength.

We start by providing background knowledge and reviewing related work in Section 2. While Section 3 presents the mentioned boundary graph-based approach in full detail, in Section 4 we return to robotic soccer simulation, explain how to utilize the proposed approach for the dribble action use case and present corresponding empirical findings.

## 2 Background and Related Work

In the following, we outline the basics that are needed to understand our approach as well as the application use case it is intended for and discuss relevant related work.

### 2.1 Robotic Soccer Simulation

In RoboCup's 2D Simulation League, two teams of simulated soccer-playing agents compete against one another using the Soccer Server [12] as real-time soccer simulation system. The Soccer Server allows autonomous software agents to play soccer in a client/server-based style: It simulates the playing field, communication, the environment and its dynamics, while the player clients connect to the server and send their intended actions (e.g. a parameterized kick or dash command) once per simulation cycle to the server. The server takes all agents' actions into account, computes the subsequent world state and provides all agents with information about their environment.

So, decision making must be performed in real-time or, more precisely, in discrete time steps: Every 100ms the agents can execute a low-level action and the world-state will change based on the individual actions of all players. Speaking about low-level actions, we stress that these actions themselves are "parameterized basic actions" and the agent can execute only one of them per time step:

- $dash(x, \alpha)$ – lets the agent accelerate by relative power $x \in [0, 100]$ into direction $\alpha \in (-180, 180]$ relative to its body orientation
- $turn(\alpha)$ – turn the body by $\alpha \in (-180, 180]$ where, however, the Soccer Server reduces $\alpha$ depending on the player's current velocity (inertia moment)
- $kick(x, \alpha)$ – kick of the ball (only, if the ball is within the player's kick range) by relative power $x \in [0, 100]$ into direction $\alpha \in (-180, 180]$
- There exist a few further actions (like tackling, playing foul, or, for the goal keeper, catching the ball) whose exact description is beyond scope.

It is clear that these basic actions must be combined cleverly in consecutive time steps in order to create "higher-level actions" like intercepting balls, playing passes, marking players, or doing dribblings.

### 2.2 Related Work on Opponent Modeling

Opponent modeling enables the prediction of future actions of the opponent. In doing so, it also allows for adapting one's own behavior accordingly. Instance-based approaches have frequently been used as a technique for opponent modeling in multi-agent games [5], including the domain of robotic soccer [2, 6].

In [15], the authors make their simulated soccer agents recognize currently executed higher-lever behaviors of the ball leading opponent. These include passing, dribbling, goal-kicking and clearing. These higher-level behaviors correspond to action sequences that are executed over a dozen or more time steps. The authors of [14] deal with the instance-based recognition of skills (shoot-on-goal skill) executed by an opponent soccer player, focusing on the adjustment of the distance metrics employed. In [8] we argued that opponent modeling is useful for counteracting adversary agents, but that we disagree with the authors of [14] claiming that "in a complex domain such as RoboCup it is infeasible to predict an agent's behavior in terms of primitive actions". Instead we have shown prototypically in [8] that a low-level action prediction can be achieved during an on-going

play using instance-based methods. We grasp this prior work of ourselves now, addressing the crucial point that we omitted to handle in that paper: Instance-based learning algorithms learn by remembering instances, which is why, for certain applications, specifically data intensive ones, retrieval times over the set of stored instances can quickly become the system's bottleneck. This issue is a showstopper in a real-time application domain like robotic soccer simulation.

### 2.3 Related Work on Index Structures for Efficient Retrieval

Index structures in instance-based methods are supposed to more efficiently guide the search for similar instances. Before the actual retrieval utilizing an index structure can take place that structure must be created. Tree-based structures have often been employed to speed up access to large datasets (e.g. geometric near-neighbor access trees [4] or nearest vector trees [10]). Tree-based algorithms that also feature online insertion capabilities include cover trees [3], boundary trees [11] (see below), or kd-trees [16] where the latter have the advantage of not requiring full distance calculations at tree nodes.

*Boundary Trees* [11] are a powerful tree-based index structure for distance-based search. They consist of nodes representing training instances connected by edges such that any pair of parent and child node belongs to different classes[1]. This fact is eponymous as with each edge traversal a decision boundary is crossed.

Given a boundary tree $\mathcal{T}$ and a new query $q$, the tree is traversed from its root by calculating the distance between $q$ and all children of the current node, moving to and traversing successively that child which has the lowest distance to $q$. Boundary trees use a parameter $k \in [1, \infty]$ that determines the maximal number of children any node is permitted to have. The retrieval is finished, if a leaf has been reached or if the current (inner) node $v$ has less than $k$ children and the distance between $q$ and $v$ is smaller than the distance between $q$ and all children of $v$. This way, a "locally closest" instance $x^*$ to the query is found, meaning that neither the parent(s) of $x^*$ nor the children of $x^*$ are more similar.

The tree creation procedure for boundary trees is inspired by the classical IB2 algorithm [1]. The next training instance $x_i$ is used as query using the so far existing boundary tree $\mathcal{T}_{i-1}$. If the result of the tree-based retrieval returns an instance $x^*$ whose class label does not match the class label of $x_i$ (i.e. $x_i$ could not be "solved" using $\mathcal{T}_{i-1}$), then $x_i$ is added as a new child node of $x^*$.

In [11], Mathy et al. propose to extend the described approach to an ensemble of boundary trees, which they name a boundary forest (BF). Essentially, they train an ensemble of (in that paper usually 50) boundary trees on shuffled versions of the training data set and employ different kinds of voting mechanisms (e.g. majority voting or Shepard weighted average [13]) using the retrieval results of the boundary trees. The Boundary Graph approach we are presenting in the next section takes some inspiration from boundary trees which is why we also use them as a reference method in our empirical evaluations.

[1] While the definition given here focuses on classification tasks, a straightforward generalization to other tasks like regression or mere retrieval can easily be made.

## 3 Boundary Graphs

It is our goal to develop an instance-based technique that covers both, a method to decide which instances to store in memory and which not as well as algorithms to build up and employ an index structure that facilitates an efficient retrieval. Boundary graphs (BG) in combination with techniques to create and utilize them, represent the backbone of our approach.

### 3.1 Notation

In what follows, we assume that each instance $x \in \mathbb{R}^D$ is a $D$-dimensional tuple of real values and has a label $l(x) \in \mathcal{L} \subset \mathbb{R}^m$ attached to it (where in case of a classification task $\mathcal{L}$ is simply the enumeration of class labels). Distance between instances is measured using a distance metric $d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}^+$ that for any two instances returns a non-negative real number $d(x, y)$. Note that we do not impose any further requirements on $d$ throughout the rest of the paper, except that, for ease of presentation, we assume it to be symmetric. Furthermore, we need a metric function $d_l : \mathcal{L} \times \mathcal{L} \to \mathbb{R}^+$ to assess the difference of label vectors.

For a given set of training instances $\mathcal{X} = \{x_1, \ldots, x_n\}$, a *Boundary Graph* $\mathcal{B} = (V, E)$ is an undirected graph without loops with a set of nodes $V \subseteq \mathcal{X}$ and a set of edges

$$E \subseteq \{(x_i, x_j) | x_i, x_j \in V \text{ and } i \neq j\}, \tag{1}$$

where, by construction, each edge from $E$ connects only instances with differing labels. This means, for each $(x_i, x_j) \in E$ it holds

$$d_l(l(x_i), l(x_j)) > \varepsilon \tag{2}$$

where $\varepsilon > 0$ is a threshold that defines when two label vectors are considered to be different. The definition given so far and the relations in Formula 1 and 2 are not finalized, most specifically since Equation 1 gives just a subset specification. We are going to concretize this specification in the next paragraphs, emphasizing upfront that the boundary graphs we are creating will be a sparse representation of the case data and, thus, contain only a tiny fraction of the edges that would be allowed to be contained in $E$ according to Equations 1 and 2.

### 3.2 Querying a Boundary Graph

Given a query $q \in R^D$ and a boundary graph $\mathcal{B} = (V, E)$, the retrieval algorithm moves *repeatedly* through the graph structure, calculating the distance between $q$ and the current node $x \in V$ as well as between $q$ and the neighbors of $x$, i.e. for all $v \in V$ for which an edge $(x, v) \in E$ exists. It successively and greedily "moves" onwards to the node with the lowest distance to $q$ until some minimum $x^\star$ has been reached, which means that $d(q, x^\star) \leq d(q, x) \forall (x^\star, x) \in E$.

Importantly, this procedure is repeated for $r$ times, where the starting node is selected randomly from $V$ each time. Hence, $r$ determines the number of random

BG_Predict($q$, $\mathcal{B}$, $r$)
**Input:** query $q \in \mathbb{R}^D$,
boundary graph $\mathcal{B} = (V, E)$,
number $r$ of random retrieval restarts,
amalgamation function $\mathcal{A}$
**Output:** BG-based prediction $\mathcal{R}(q)$

```
// retrieval
N_q^r ← BG_Retrieve(q, B, r)
// prediction (cf. Eqn. 4-6)
R(q) ← A(N_q^r)
return R(q)
```

BG_Retrieve($q$, $\mathcal{B}$, $r$)
**Input:** query $q \in \mathbb{R}^D$,
boundary graph $\mathcal{B} = (V, E)$ with $V \neq \emptyset$,
number $r$ of random retrieval restarts
**Output:** $r$-dimensional vector $\mathcal{N}_q^r$ of potential nearest neighbors

```
N_q^r ← r-dimensional vector
for i = 1 to r do
    x* ← random node from V
    stop ← false
    while stop = false do
        x ← arg min_{v∈V s.t. (x*,v)∈E} d(q, v)
        if d(q, x) < d(q, x*)
        then x* ← x else stop ← true
    N_q^r[i] ← x*
return N_q^r
```

**Algorithm 1:** Boundary Graph-Based Prediction and Retrieval

retrieval starting points from which the distance-guided search is initiated. Consequently, as retrieval result a vector $\mathcal{N}_q^r = (n_1, \ldots, n_r)$ of $r$ estimated nearest neighbors is obtained.

Algorithmically, we embed the delineated step (function BG_Retrieve in Algorithm 1) into the superjacent function BG_Predict for boundary graph-based prediction which, effectively, performs both, the retrieval task and the prediction on top of it. The entries of the vector of $r$ nearest neighbor estimates are combined to form an overall prediction $\mathcal{R}(q)$ using some amalgamation function $\mathcal{A}$, such that

$$\mathcal{R}(q) = \mathcal{A}(\mathcal{N}_q^r) = \mathcal{A}((n_1, \ldots, n_r)). \tag{3}$$

For classification tasks, we might use a simple majority vote

$$\mathcal{A}((n_1, \ldots, n_r)) \in \arg\max_{t \in \mathcal{L}} |\{n_j | l(n_j) = t, j = 1, \ldots, r\}| \tag{4}$$

or an inverted distance-weighted voting scheme, like

$$\mathcal{A}((n_1, \ldots, n_r)) \in \arg\max_{t \in \mathcal{L}} \sum_{j=1}^{r} \begin{cases} 1/d(q, n_j) & \text{if } l(n_j) = t \\ 0 & \text{else} \end{cases}. \tag{5}$$

In a similar manner, for regression tasks the estimated value becomes [13]

$$\mathcal{A}((n_1, \ldots, n_r)) = \frac{\sum_{j=1}^{r} l(n_j)/d(n_j, q)}{\sum_{j=1}^{r} 1/d(n_j, q)}. \tag{6}$$

A pseudo-code summary of the entire retrieval and preciction approach using a BG is given in Algorithm 1. For the empirical case study presented below we stick to a simple majority vote according to Equation 4 and employ a normalized $L_1$ norm as distance measure $d$. Before, however, we can utilize a BG, we must build it up which is why we focus on the construction of boundary graphs next.

BG_Construct($\mathcal{X}$, $r$)
**Input:** set of train instances $\mathcal{X} = \{x_1, \ldots, x_n\}$, number $r$ of random retrieval restarts
**Output:** boundary tree $\mathcal{B}$
1: $\mathcal{B} \leftarrow (\emptyset, \emptyset)$
2: *// loop over all instances*
3: **for** $i = 1$ **to** $n$ **do**
4: $\quad \mathcal{B} \leftarrow$ BG_Train($\mathcal{B}, x_i, r$)
5: **return** $\mathcal{B}$

BG_Train($\mathcal{B}$, $x$, $r$)
**Input:** single (new) instance $x$, boundary graph $\mathcal{B} = (V, E)$, number $r$ of random retrieval restarts
**Requires (global variables):** amalgamation function $\mathcal{A}$, metric $d$ and $d_l$, label discrimination threshold $\varepsilon$
**Output:** (possibly extended) boundary graph $\mathcal{B}$
1: **if** $V = \emptyset$ **then** $V \leftarrow V \cup x$
2: **else**
3: $\quad \mathcal{N}_x^r \leftarrow$ BG_Retrieve($x, \mathcal{B}, r$)
4: $\quad$ **for** $i = 1$ **to** $r$ **do**
5: $\quad\quad \delta \leftarrow d_l(l(x), l(\mathcal{N}_x^r[i]))$
6: $\quad\quad$ **if** $\delta > \varepsilon$ **then**
7: $\quad\quad\quad V \leftarrow V \cup x$
8: $\quad\quad\quad E \leftarrow E \cup (x, \mathcal{N}_x^r[i])$
9: $\quad$ **return** $(V, E)$

**Algorithm 2:** Construction of and Retain Procedure for Boundary Graphs

### 3.3 Graph Construction

We assume that the instances $x_1$ to $x_n$ from the set of training instances $\mathcal{X}$ are presented to the boundary graph construction algorithm successively. Given a single training instance $x_i$, the algorithm first queries the boundary graph $\mathcal{B}_{i-1} = (V_{i-1}, E_{i-1})$ which has been trained for the preceding $i-1$ training instances, yielding a vector $\mathcal{N}_{x_i}^r = (n_1, \ldots, n_r)$ of $r$ possible nearest neighbors. The algorithm then iterates over these $n_j$ $(j = 1, \ldots, r)$ and, if $d_l(l(x_i), l(n_j)) > \varepsilon$ (i.e. $n_j$ does "not solve" $x_i$, which in the case of classification tasks boils down to $l(x_i) \neq l(n_j)$), then $x_i$ is added as a new node to $V_{i-1}$ and a (bidirectional) edge $(x_i, n_j)$ is added to $E_{i-1}$. The resulting, extended boundary graph is accordingly denoted as $\mathcal{B}_i$. To sum up, training instances are added as nodes to the graph (including connecting edge), if the algorithm stochastically discovers a random retrieval starting point for which the currently existing boundary graph's prediction would be wrong and where, hence, a correction is needed.

Again, a pseudo-code summary of the algorithm to constructively building up a boundary graph for a sequence of training instances $\mathcal{X}$ is provided in Algorithm 2, denoted as BG_Construct. Note that the algorithm can be easily deployed in an online setting where new instances arrive during runtime by simply calling the BG_Train function given in the right part of Algorithm 2. Additionally, Figure 1 visualizes exemplary boundary graphs for two synthetic two-dimensional two-class problem.

*Constructing vs. Applying the Graph Structure* As we will show below, the algorithms described have a logarithmic retrieval complexity in the amount of training instances $n$ for the opponent action prediction dataset we use subsequently. Accordingly, training time scales mildly as well because each train step

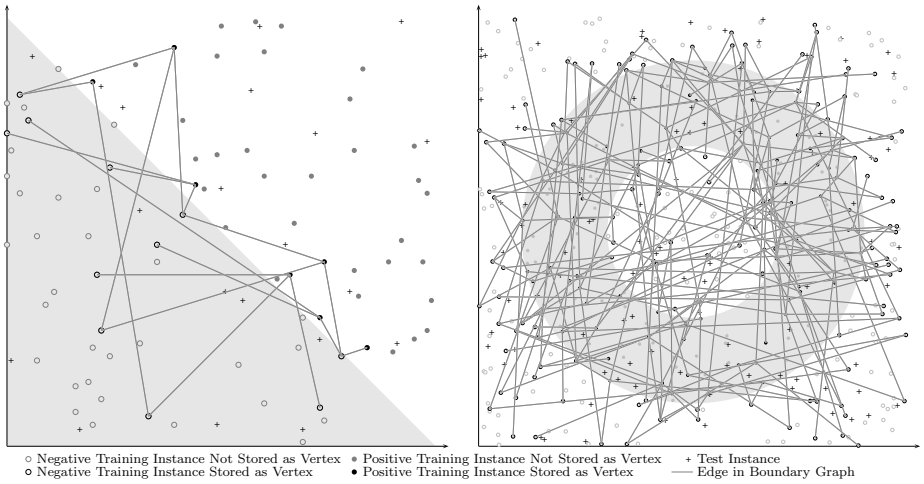


**Fig. 1.** Two Exemplary Visualizations of Boundary Graphs for Synthetic Domains: Out of the 80 training instances (left), 19 are included in the graph's set of vertices (11 from the negative class, 8 from the positive one), when trained using $r = 9$. Among that set of nodes, there are 88 possible edges that would cross the decision boundary. From those, 40 are included in the graph's set of edges. The boundary graph for the "doughnut" domain (right) has been constructed using $r = 3$. The graph stores 203 out of the 400 training instances and connects them by 534 edges.

essentially includes a retrieve step, rendering the complexity of training to be a loop of $n$ repetitions wrapped around the retrieval procedure ($O(n \log n)$).

The boundary graph approach has the favorable characteristic to be an *anytime retrieval algorithm*. By handling the parameter $r$ of random retrieval starting points within the graph differently during training ($r_t$) and the application ($r_a$) of the learned graph, i.e. separating $r_t := r$ from $r_a$ ($r_a \neq r$), one can gain a significant performance boost by letting $r_a > r_t$, given that a sufficient amount of time is available for the system to respond to a query. This is a desirable property in real-time and online application settings since the accuracy of the retrieval grows with $r_a$ as we will delineate in the next section.

## 4 Empirical Evaluation

Our empirical investigations on the boundary graph approach were primarily driven by our target application problem of predicting an opponent soccer player's next low-level action. We focus on a dribbling opponent, leaving the investigation of other opponent behaviors for future work. We first more introduce the task at hand more precisely and then present achieved classification results including an analysis of our algorithms' scaling behavior. In a separate paper [9], we present detailed results on the performance of our proposed algorithms for a variety of classical benchmark datasets beyond the realm of robotic soccer.

### 4.1 Problem Formulation and Data Collection

We focus on the task of predicting a ball leading opponent player's next dribbling action, although we stress that our approach could generally be applied to any other action prediction task as well. Therefore, we selected an opponent agent, placed it randomly on the field with the ball in its kick range and allowed it to dribble towards our goal for maximally 20 consecutive time step (or till it lost the ball) when it was relocated to a new random position. The state of the dribbler was described by a 9-tuple consisting of the player's and the ball's position on the field (4), the player's and ball's current velocity vectors (4) and the angle of the player's body orientation (1). The actually performed action (kick, dash, or turn) was extracted from the game log file, though we might deduce this piece of information during a match, too, by applying inverse kinematics on two consecutive states exploiting the knowledge about the physics models the Soccer Server [12] applies. We collected half a million training examples using the described methodology using a FRA-UNIted agent as dribbling opponent which utilizes its established dribbling behavior that was trained with reinforcement learning [7]. The class distribution in this dataset features 7.2% turning, 20.4% kicking, and 72.4% dashing actions such that a naive classifier that always predicts the majority class would yield an error of 27.6%. Note that in the context of the evaluation presented here, we solely focused on the classification of the *type* of the performed action, not on its real-valued parameter(s) (cf. Section 2.1).

We compare our boundary graph approach to the classical nearest neighbor algorithm (which linearly iterates over all stored training instances) as well as to the boundary forest approach (BF) from the literature (cf. Section 2.3). We measure performance in terms of the achieved classification error on an independent test set as well as in terms of required real-time (on a contemporary 3 GHz CPU, single-core, i.e. without any parallelization[2]).

### 4.2 Results

Table 1 summarizes the remaining classification errors when predicting the opponent's low-level dribble actions for different training set sizes $n$. All numbers reported are averages over 100 repetitions of the experiment using different random number seedings. As expected, the nearest neighbor classifier turns out to be a simple, but computationally prohibitive baseline. When opposing boundary forests and boundary graphs, it is advisable to compare settings that are conceptually similar, viz when the number $t$ of trees and the number $r$ of random retrieval restarts match. The result table reports results for $r = t \in \{50, 100\}$ and shows that boundary graphs slightly, but consistently outperform the forest approach except for small training set sizes where, however, all approaches "fail" since their accuracy is not so far off the error of the naive classifier (27.6%) that just predicts the majority class.

[2] We emphasize that all discussed approaches are easily parallelizable and that computation times could, thus, be reduced dramatically given the appropriate hardware.

| $n$ | $k$-NN ($k = 1$) | BF ($t = 50$) | BG ($r = 50$) | BF ($t = 100$) | BG ($r = 100$) |
|---|---|---|---|---|---|
| 400 | 27.80 ± 0.48 | **28.26** ± 0.44 | 29.41 ± 0.47 | **27.64** ± 0.45 | 29.44 ± 0.45 |
| 1600 | 23.92 ± 0.19 | **25.12** ± 0.21 | 25.22 ± 0.20 | **24.71** ± 0.21 | 25.49 ± 0.20 |
| 6400 | 22.10 ± 0.10 | 22.14 ± 0.11 | **21.88** ± 0.11 | **21.51** ± 0.11 | 21.92 ± 0.11 |
| 25600 | 18.77 ± 0.06 | 18.55 ± 0.06 | **18.07** ± 0.05 | **17.94** ± 0.05 | 18.19 ± 0.05 |
| 51200 | 16.64 ± 0.02 | 16.31 ± 0.04 | **15.72** ± 0.03 | 15.75 ± 0.03 | **15.72** ± 0.02 |
| 102400 | 13.46 ± 0.03 | 13.27 ± 0.02 | **12.62** ± 0.02 | 12.78 ± 0.03 | **12.61** ± 0.02 |
| 204800 | 8.70 ± 0.01 | 9.25 ± 0.02 | **8.77** ± 0.01 | 8.79 ± 0.01 | **8.64** ± 0.01 |
| 409600 | 5.42 ± 0.01 | 5.90 ± 0.01 | **5.43** ± 0.01 | 5.55 ± 0.01 | **5.22** ± 0.01 |

**Table 1.** Classification errors and belonging standard errors of the discussed algorithms in percent subject to different amounts of training data for 100 experiment repetitions. Better-performing algorithms (between BF and BG only) are highlighted in bold.

The left part of Figure 2 visualizes the scaling behavior of boundary graphs (black) and boundary forests (gray), reporting the average number of milliseconds required to answer a single test query, i.e. to predict the opponent's next dribble action, subject to different amounts of training data that has been processed to generate the BF/BG. Apparently, boundary graphs need about a third more computational effort compared to their same-sized tree-based counterparts, but achieve lower classification errors as discussed in the preceding paragraph. It is worth noting that we have set the value of the BF parameter $k$ (cf. Section 2.3) to infinity during all our experiments. Setting $k$ to a finite value would further reduce the computational requirements of that algorithm, but at the same time impair its performance even more as delineated by [11].

Another interesting observation is that a boundary graph ($r = 50$), which has been constructed using $n = 409.6k$ instances, stores about 60% of them as vertices in the graph (space complexity grows linearly with $n$). Yet, during a BG-based retrieval for a single test query $q$ the distance calculation (which, essentially, represents the computational bottleneck) between $q$ and stored instances must, effectively, be done for only $\approx 1.8\%$ of the $n$ given training instances.

After all, the chart shows that any of the graph- or forest-based approaches have a logarithmic time complexity and could very well be deployed practically by a soccer-playing agent since the retrieval time of less than $40ms$ (on the mentioned hardware) would fit well into a soccer simulation time step (even without any parallelization). Since the addition of a single new instance requires basically one retrieval plus a loop over $r$ (which has constant effort in $n$), it requires roughly the same amount of computation as processing a test query and, thus, even an online extension of a boundary graph during a running match is feasible, for example when observing the current opponent dribbling. By contrast, the nearest neighbor classifier (also shown in the chart) has linear complexity and requires more than $50ms$ already for 8000 stored instances (and even $3000ms$ per test query for $n = 409.6k$) which renders this algorithm practically useless.

An outstanding characteristic of the boundary graph algorithm is its anytime behavior. By increasing the number of random retrieval restarts during the application phase (for example in a match, when in a specific time step less

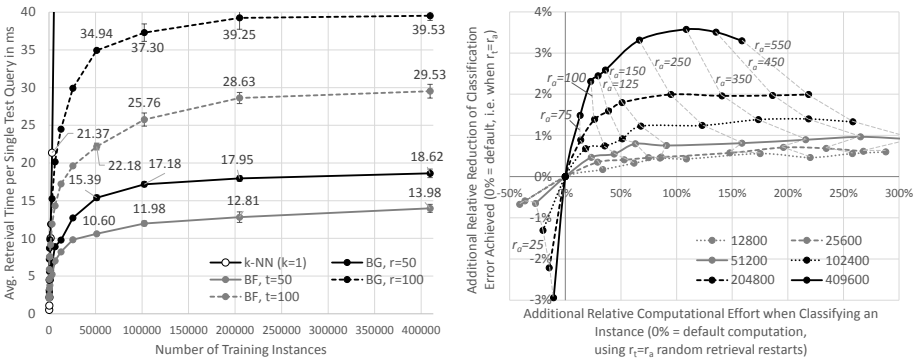


**Fig. 2.** Scaling Behavior of Boundary Graphs: See the text for details.

other computations are done for whatever reason) the accuracy of the prediction increases. This relationship is visualized in the right part of Figure 2 for a boundary graph that has been *constructed* using $r_t = 50$ random retrieval restarts. So, the point of origin represents the "default" setting where $r_t = r_a = 50$. For positive $x$ values the computational budget during application (not during training, i.e. $r_t = 50$ is not altered) has been increased expressing the *relative* extra effort on top of the default in terms of additional real-time. Likewise, negative $x$ values denote that less computational power is invested into the retrieve process during testing. The ordinate shows the impact of the described variation of $r_a$ in terms of *relative* gain/loss in classification performance compared to what is achieved with the default setting. So, for example, for $n = 409.6k$ training instances (here, processing a test query for $r_a = 50$ needs $18.6ms$ on average) we observe that by doubling the retrieval time (+100%, i.e. $37.2ms$, corresponding to $r_a \approx 350$) the originally achieved classification error can be reduced by ca. 3.5%.

## 5 Conclusion

We have proposed boundary graphs as a useful and scalable tool for instance-based prediction. Although we have focused solely on its use for classification throughout this paper, the approach is general enough to cover other tasks like regression or mere instance retrieval as well. We provided algorithms for creating and utilizing boundary graphs and applied them successfully for the prediction of the next low-level action of a dribbling simulated soccer player. In so doing, we found that this approach scales very well and is applicable under hard real-time constraints even with large sets of training data which are required for high-quality predictions. Our next steps include the employment of this approach for determining the real-valued parameters of the predicted action which, of course, represents a regression task. Moreover, we also intend to evaluate in depth the performance of boundary graphs for other established benchmark datasets beyond the realm of robotic soccer.

## References


1. Aha, D., Kibler, D., Albert, M.: Instance-Based Learning Algorithms. Machine Learning 6, 37–66 (1991)
2. Ahmadi, M., Keighobadi-Lamjiri, A., Nevisi, M., Habibi, J., Badie, K.: Using a Two-Layered Case-Based Reasoning for Prediction in Soccer Coach. In: Proceedings of the International Conference of Machine Learning; Models, Technologies and Applications (MLMTA'03). pp. 181–185. CSREA Press (2003)
3. Beygelzimer, A., Kakade, S., Langford, J.: Cover Tree for Nearest Neighbor. In: Proceedings of the Twenti-Third International Conference on Machine Learning (ICML). pp. 97–104. ACM Press, Pittsburgh, USA (2006)
4. Brin, S.: Near Neighbors Search in Large Metric Spaces. In: Proceedings of the Twenty-First International Conference on Very Large Data Bases (VLDB). pp. 574–584. Morgan Kaufmann, Zurich, Switzerland (1995)
5. Denzinger, J., Hamdan, J.: Improving Modeling of Other Agents Using Stereotypes and Compactification of Observations. In: Proc. of 3rd International Conference on Autonomous Agents and Multiagent Systems. pp. 1414–1415. New York (2004)
6. Fukushima, T., Nakashima, T., Akiyama, H.: Online Opponent Formation Identification Based on Position Information. In: H. Akiyama, O. Obst and C. Sammut and F. Tonidandel, editors, RoboCup 2017: Robot World Cup XXI, LNCS. pp. 241–251. Springer, Nagoya, Japan (2017)
7. Gabel, T., Breuer, S., Roser, C., Berneburg, R., Godehardt, E.: FRA-UNIted – Team Description 2017 (2018), Supplementary material to RoboCup 2017: Robot Soccer World Cup XXI
8. Gabel, T., Godehardt, E.: I Know What You're Doing: A Case Study on Case-Based Opponent Modeling and Low-Level Action Prediction. In: Proceedings of the Workshop on Case-Based Agents at the International Conference on Case-Based Reasoning (ICCBR-CBA 2015). pp. 13–22. Frankfurt, Germany (2015)
9. Gabel, T., Sommer, F.: Case-Based Learning and Reasoning Using Layered Boundary Multigraphs. In: Proceedings of the International Conference on Case-Based Reasoning (ICCBR 2022). Springer, Nancy, France (2022)
10. Lejsek, H., Jonsson, B., Amsaleg, L.: NV-Tree: Nearest Neighbors in the Billion Scale. In: Proceedings of the First ACM International Conference on Multimedia Retrieval (ICMR). pp. 57–64. ACM Press, Trento, Italy (2011)
11. Mathy, C., Derbinsky, N., Bento, J., Rosenthal, J., Yedidia, J.: The Boundary Forest Algorithm for Online Supervised and Unsupervised Learning. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence. pp. 2864–2870. AAAI Press, Austin, USA (2015)
12. Noda, I., Matsubara, H., Hiraki, K., Frank, I.: Soccer Server: A Tool for Research on Multi-Agent Systems. Applied Artificial Intelligence 12(2-3), 233–250 (1998)
13. Shepard, D.: A 2Dimensional Interpolation Function for Irregularly-Spaced Data. In: Proc. of the 23rd ACM National Conference. pp. 517–524. ACM (1968)
14. Steffens, T.: Similarity-Based Opponent Modelling Using Imperfect Domain Theories. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG05). pp. 285–291. Colchester, United Kingdom (2005)
15. Wendler, J., Bach, J.: Recognizing and Predicting Agent Behavior with Case-Based Reasoning. In: D. Polani and A. Bonarini and B. Browning (editors), RoboCup 2003: Robot Soccer World Cup VII. pp. 729–728. Padova, Italy (2004)
16. Wess, S., Althoff, K., Derwand, G.: Using k-d Trees to Improve the Retrieval Step in Case-Based Reasoning. In: Proceedings of the 1st European Workshop on Case-Based Reasoning (EWCBR 1993). pp. 167–181. Springer, Germany (1993)